Simulating Distributed Systems

Harvey B. Newman, Iosif C. Legrand

Charles C. Lauristsen Laboratory of High Energy Physics
California Institute of Technology, Pasadena, CA 91125, USA

Abstract. The simulation framework developed within the "Models of Networked Analysis at Regional Centers" (MONARC) project as a design and optimization tool for large scale distributed systems is presented. The goals are to provide a realistic simulation of distributed computing systems, customized for specific physics data processing tasks and to offer a flexible and dynamic environment to evaluate the performance of a range of possible distributed computing architectures. A detailed simulation of a large system, the CMS High Level Trigger (HLT) production farm, is also presented.

INTRODUCTION

The design and optimisation of large scale distributed systems, requires a realistic description and modelling of the data access patterns, the data flow across the local and wide area networks, and the scheduling and workload presented by hundreds of jobs running concurrently on large scale distributed systems exchanging very large amounts of data.

A process-oriented approach for discrete event simulation has been adopted because it is well suited to describe various activities running concurrently, as well as the stochastic arrival patterns typical of this class of simulations [1]. This simulation program is based on Java(TM) technology because of the support for the necessary methods and techniques needed to develop an efficient and flexible simulation frame [2].

DESIGN CONSIDERATIONS

The process-oriented approach for discrete event simulation is based on Threaded objects, or "Active Objects" (having an execution thread, program counter, stack, mutual exclusion mechanism...). They offer great flexibility in simulating the complex behaviour of distributed data processing programs. This approach offers a natural way of describing complex running programs that are data dependent and which concurrently compete for shared resources.

The MONARC simulation program [1] is built with Java(TM) technology. Java has built-in multi-thread support for concurrent processing, which can be used for simulation purposes by providing a dedicated scheduling mechanism. Java also offers good support for distributed objects architectures and for graphics. The flexible graphics tools, and facilities to analyse data interactively, are essential in any simulation project.

The tool's "simulation engine" provides a dedicated scheduling mechanism that is based on semaphores for the "Active Objects". It also provides a mechanism to dynamically add or remove objects from the system. Handling dynamically loadable modules is essential to describe complex configurations, which may change or evolve in time. The "Active Object" is the basic class that must be inherited by all the entities in the simulation, which require a time dependent behaviour. It provides the methods for synchronous and asynchronous communications with other objects, and the mechanism to communicate with the simulation engine so that it can be interrupted, suspended and resumed during execution. Objects, which extend this basic class, may implement any specific time dependent behaviour, which can be a function of

CP583, *Advanced Computing and Analysis Techniques in Physics Research: VII International Workshop*
edited by P. C. Bhat and M. Kasemann

messages or data received, its previous state(s), and its access to certain shared resources. In this way it is possible to implement highly non-linear processes such as caching and swapping. It also offers a means of describing the stochastic input pattern for jobs and activities in the system.

Shared resources, like CPU or I/O links, are represented in the simulation as normal objects, but access to their different update methods needs to be made, synchronised with the external "running" entities. There is a mutual exclusion mechanism when accessing unique atomic parts that avoids interruption: this guarantees the correct representation of the execution of concurrent processes.

As the number of jobs necessary to be simulated in such applications may be huge, a dedicated structure that allows "Active Objects" recycling was implemented to improve the simulation efficiency. The interrupt mechanism, implemented as an atomic (synchronised) self addressed event, for the "Active Objects" offers an effective way to simulate discrete event processes assuming a "continuous" flow in time between events which modify parts of the system. The interrupt functionality offers the possibility to simulate efficiently and accurately concurrent processes that need to share resources. As an example, the way multitasking is simulated is presented in Figure 1.

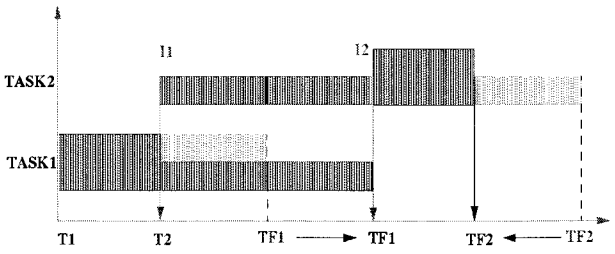

Figure 1. Modelling multitasking processing based on an "interrupt" scheme.

Referring to this figure, when a first job (Task1) starts, the time it takes is evaluated (original TF1), and this "Active" object enters into a wait state for this amount of time unless it is interrupted. If a new job (Task2) starts on the same hardware, it will cause an interrupt to the first task. Both tasks will share the same CPU power and the time to complete for each of them is re-computed assuming that they share the CPU equally or based on a running priority scheme (new TF1 and original TF2). Then both jobs will enter into a wait state and listen for other interrupts. When the first job (Task1) is finished, it creates another interrupt to re-distribute the resources for the remaining jobs. This model assumes that resource sharing is maintained between any discrete events (e.g. new job submission, job completion) that occur during the simulated time interval.

The simulation program includes a convenient set of interactive graphical tools allowing dynamic configurations as well as presentation and analysis of results. It provides a powerful development tool for evaluating and designing large scale distributed systems.

Simulation of the CMS ORCA-HLT Production Farm, Spring 2000

The simulation program was tested and validated with specific Queuing Theory problems as well as with a set of dedicated test bed measurements [3]. The simulation of a large production farm, based on Object Oriented database is presented here.

The immediate goal of the Spring ORCA/HLT production was to prepare a fully digitised sample of 2 million events with full pile-up at a simulated luminosity of 10^{34} cm^2s^{-1}. This is a multi-stage process taking 2 million CPU minutes of computing and involving the transport of some 70 Tera-bytes of pile-up hits. The production starts from 2TB GEANT3 fz files, which are converted into 2TB Objectivity/DB format and stored in the HPSS system. The final results are again stored in an Objectivity/DB where they are used in reconstruction and analysis studies. This was achieved making use of a farm of 200 high performance commodity PC's running the Linux operating system. The Data Server (AMS) of Objectivity/DB was found to be particularly powerful in allowing new farm architectures to be studied. The Central Data Recording (CDR) system of IT/PDP was used to migrate data safely into HPSS, from which it is is now being accessed by Physicists studying the HLT.

The real set-up and realistic values for the hardware configuration parameters were used for the configuration of the simulation. For each type of data processing jobs the amount of I/O per event and the effective CPU per event were considered as input parameters for the simulation. The way programs read the events and all the pile-up structures was included in the simulation.

The simulation results are in good agreement with the measurements done during this CMS production. As an example, the distribution of the total time per job for the Muon Production task is compared with the measured values in Figure 2.

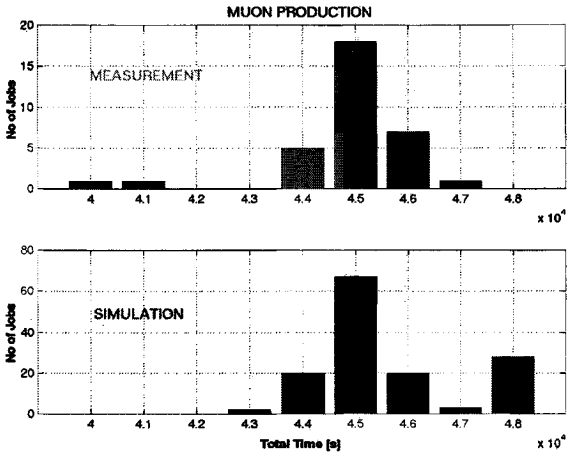

Figure 2. Comparison between the measurements and the simulation of the total time distribution for the "Muon Production" jobs.

The distribution of the job's efficiency for both types of production jobs is presented in Figure 3.

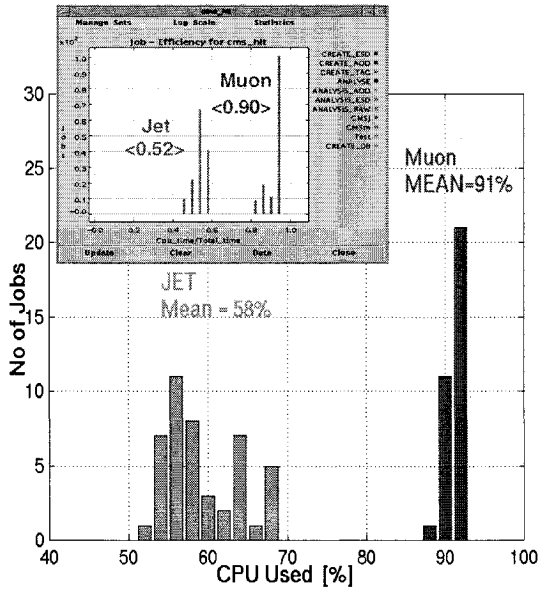

Figure 3. The distribution of the job's efficiency for the "Jet" and "Muon" production tasks.

Modelling correctly and understanding current production data processing farms is essential for the future design of the large-scale distributed systems.

SUMMARY

A CPU and code-efficient simulation approach to the problem of simulation of distributed computing systems has been developed and tested within the MONARC Collaboration. It provides a transparent way to map the distributed data processing, data transport and analysis tasks onto the simulation frame, and can describe dynamically even very complex computing models.

ACKNOWLEDGEMENTS

This work has been performed in collaboration with the MONARC project at CERN. We would like to thank David Stickland and Tony Wildish for the help in understanding the CMS production farm configuration and set-up.

REFERENCES

1. I.C. Legrand, H.B. Newman, "The Monarc Toolset for Simulating Large Network-Distributed processing Systems", Proc. of the 2000 Winter Simulation Conference. And MONARC Simulation Tool http://www.cern.ch/ MONARC/sim_tool/

2. PLOTEMY II "Heterogeneous concurrent Modeling in Java" http://www.eecs.berkeley.edu

3. Y. Morita et al. "Validation of Monarc Simulation Tools", CHEP2000, Padua, Italy (http://chep2000.pd.infn.it/, paper number 113